WART Interpreter Operation Guide

The WART Interpreter is an interactive string manipulation engine. It is designed to allow rapid, repeatable format conversion for various text data files. It allows direct input of data from disk or web files, data extraction and formatting, and output to disk or web destinations.

Overview

The interpreter uses a single page editor window in which WART Language commands are entered. A WART program will typically consist of a few lines of source code. The commands for many related programs are typically kept available in the editing window where they can be easily copied and revised. These commands are saved from one session to the next, and each short "program" may be individually selected for execution. This contrasts to the traditional programming style which uses a one-program-per-source-file or one-program-per-many-source-files approach.

The WART program text is stored in the file **`Wart.ini`** in the same directory from which the interpreter is run.

The WART interpreter may be executed from the command line. Parameters name the source file, identify the function within the file and pass a parameter string to that function.

```
        WART.EXE <SourceFileName> <FunctionName> <ParameterString>
```

The WART Language consists of a sequence of tokens that are converted into an intermediate P-code which is used by the execution engine. The tokens represent keywords, variable identifiers and literals. Keywords may represent statements followed by parameters or they may be symbols that perform flow control. All statements begin with a keyword. The parameter list for the statement may consist of identifiers or literals. There is no concept of expressions or compound statements within the language syntax, although strings may contain arithmetic expressions which can be evaluated to yield signed integer results, nested loops are allowed and subprograms may be defined and invoked.

Variables

All variable identifiers correspond to variable-length character strings. Certain statements perform arithmetic operations on variables. This is accomplished by converting text to 32-bit signed integers, performing the operation, then converting back to a text string representation.

Internally all strings are literal representations of the files or text. For display purposes and data entry (as in typing in the literal quoted strings) an escaped format is used. The escaped form is a modification of the traditional C language version that uses the "\" character and following special symbol groups:

\n	CR/LF pair	\r	CR
\\	\	\l	LF
\xx	hex characters 00-1F	\Q	"
\xx	hex characters 7F-FF	\q	'

Note the (annoying) requirement that you must use "\\" when entering the backslash in windows disk file names. A convenient feature allows the use of the '/' in place of the windows standard '\' in disk file names in the LOAD and STORE operators.

Frequently strings will represent a list of items such as numeric values or file names. The representation will be a variable-length character string with a defined delimiter separating list elements. Some lists consist of a set of FieldName=FieldValue pairs.

A key concept is that these "variable-length character strings" can be the entire contents of disk files.

```
        LOAD X "http://dilbert.com/comics/dilbert/archive/";
```

loads a complete HTML file into a string variable called X.

```
        LOAD    X     "http://dilbert.com/comics/dilbert/archive/images/dilbert20024386030101.gif";
```

loads a GIF file into the string variable called X.

There is no practical limit to the size of a string variable so loading multi-megabyte files is not a problem. Copying a string to disk is equally simple:

```
        STORE "dilbert20020101.gif" X;
```

Numeric values are generally interpreted as decimal integers and are normally 32-bit signed values. Commas may be used as thousands separators and are ignored on input. Non-numeric characters may be used as delimiters for lists of numeric values.

Flow Control

Flow control consists of bracketing symbols that form unconditional loops, and conditionals that exit loops. The loop forms a reverse branch and the conditional structure performs forward branches. The intention of this design is to make an absolutely minimal language implementation that is capable of handling such concepts as "For Each File" (from a list) perform some action, or "For Every Occurrence" (of a pattern in a file) perform some action.

Many statements may "fail", based on the operands they are given. Normally, failure causes the immediately enclosing loop to be terminated. However, an optional "else" portion of the loop, indicated by the "|" (vertical bar), may be used to handle such failure conditions.

Consider the Algol conditionals and loops:

```
        if x = y then statement;
        if (123 <= x) and (x <= 234) then statement1 else statement2 ;
        for i:=0 to 10 do statement;
```

The equivalent constructs would be:

```
        [ range x y y; statement; exit ]
        [ range x 123 234; statement1; exit | statement2; exit ]
        set i 0; [ range i 0 10; statement; ADD i 1; ]
```

Note the inherent looping behavior of bracketed sections of code. If straight-through operation is desired the **exit** statement must be included. Although single statements are shown these could actually be any sequence of statements, including other loops. Equivalent Algol notation would require compound statements using **begin** ... **end**.

Statements

Specialized statements perform string searches, field extraction, concatenation, I/O, and certain application-specific operations related to map generation.

Reserved Words

The following keywords are defined:

Statement	Description
; (Semicolon)	Statement separator.
[(Left Bracket)	Beginning of an execution loop.
] (Right Bracket)	End of loop block. Means unconditional GoTo the top of the loop.
| (Vertical Bar)	Equivalent to an ELSE in the current loop block.
ADD res inc	Increment a numeric string by a given amount.

Statement	Description
CALL name param	Invoke the FUNCTION by name. Param becomes the "*" on entry to the function. The return value of the function is in "*".
EVAL expr	Evaluate the arithmetic expression into a numeric result value. The result replaces the expression. Variable substitution is supported.
EXEC commandline	Execute the commandline as a DOS shell.
EXIT	Break out of the current loop, i.e. GoTo the matching "|" or "]".
FIELD str delim num	str is a list delimited by delim. Return the num'th field in "*".
FORMAT var codes	var is modified to match the formatting codes.
FUNCTION name	identifies the beginning of a set of statements
GREP string Lpatt Rpatt	Search for text between two given delimiters. The value of string is updated. Result is placed in "*". **Failing to find the delimited string will exit the loop.**
INPUT	Wait for user keyboard input. Text entry appears in $KEYBOARD variable. "Enter" continues execution. **"Escape" exits the current loop.**
LOAD res filename	Read a disk or HTTP or FTP file into a string. The actual filename is placed in "*". The filename variable may be a \n-separated list of names. The filename is updated by removing the name. HTTP files also place results in $HEADER and $BODY. **Failing to read the file will exit the current loop.**
MAPINIT parameters	Create a blank bitmap to draw on and give scale parameters.
MAPLINE coords width color	Draw a line in a given color and style on a map.
MAPMARK coords mark color	Place markers at specific coordinates on a map.
MAPSAVE filename	Write the bitmap to a disk file.
MAPTOUR res coords limits	res becomes an ordered list of points from coords that meet specified Traveling Salesman limits. Coords is updated to delete the points.
MINMAX res values	Take the list of values and compute I, Min, Sum, Ctr, Avg, Span, Max
OPEN res	Display an Open File dialog and allow selection of one or more files. Res becomes \n delimited list of filenames. **User cancel of the dialog is a failure which exits the current loop.**
POST URL text	Post the text to the specified URL. Results will be placed in "*" and in $HEADER and $BODY.
PRINT text	text is ASCII with \n line separators. Print the text on the default printer.

Statement	Description
RANGE value lowlim highlim	Verify that a text or numeric string falls within a certain range. **If the value does not fall within the specified range the statement fails and the current loop is exited.**
REP str old new	str is modified by replacing all occurrences of the string old with new.
SELECT ID XML pattern	Select a field from an XML file. ID is a list of tags which identify the required field which will be returned in "*". The field must contain the specified pattern. The XML value will be updated by delting text up through the returned field. **If the pattern is not found it is a failure which exits the current loop.**
SET res str str str ...	Concatenate zero or more strings to form a new string value.
SHOW str	Display the parameter as a status value.
STORE filename str	Write a string to a disk or web file.
SUBSTR str patt resA resB ...	Extract multiple substring fields from str based on a pattern patt. Results in up to 26 variables resA ... ResZ.
TAGS TagList XMLtext	Validate an XML file in XMLtext and list all the tags that it contains. Result in TagList **If validation fails the current loop is exited.**
UTM res str	Convert str "Zone Northing Easting" to "Lat Lon" and vice versa. Result is placed in res.

The system uses certain pre-defined variable identifiers in various operations.

Identifier	Description
$BODY	Body of a HTML file loaded from the web
$FTPpass	Password to be used for FTP connections.
$FTPuser	User name to be used for FTP connections
$HEADER	HTML header of a file loaded from the web
$KEYBOARD	Data entered from the keyboard
$NOW	Current time
$STATUS	Status message displayed
$TIMESTAMP	Date and Time that this program started execution.
$UTC	Current time in UTC
$VERSION	Version name, number and date of the WART Interpreter
***** (Asterisk)	Result of field extraction operations

Statement Reference

; (Semicolon)
[(Left Bracket)
] (Right Bracket)
| (Vertical Bar)
ADD res inc
CALL name param
EVAL expr
EXEC commandline
EXIT
FIELD str delim num
FORMAT var codes
FUNCTION name
GREP string Lpatt Rpatt
INPUT
LOAD res filename
MAPINIT parameters
MAPLINE coords width color

MAPMARK coords mark color
MAPSAVE filename
MAPTOUR res coords limits
MINMAX res values
OPEN res
POST URL text
PRINT text
RANGE value lowlim highlim
REP str old new
SELECT ID XML pattern
SET res str str str ...
SHOW str
STORE filename str
SUBSTR str patt resA resB ...
TAGS TagList XMLtext
UTM res str

; (Semicolon) **Statement Separator**

The semicolon is used as a visual separator for the convenience of the user. All statements begin with a key word and the keywords are not allowed as parameters to statements. Thus, a separator is not required by the syntax.

! (Exclamation Point) **End-of-Line Comment**

Documentation and comments are denoted by the Exclamation Point.

```
SET x "ABRACADABRA";       ! Set variable x to a magic word.
```

[(Left Bracket) **Beginning of loop**

] (Right Bracket) **End of loop**

Brackets are used to form a structured loop construct. Loops themselves are unconditional "Forever" loops.

Conditionals are implemented with the concept of "Statement Failure". For example, the failure of a GREP to find a required string will cause a loop enclosing the GREP to be exited.

```
SET x "ABRACADABRA";
[
       GREP x "A" "R";
       SET out out * "\n";        ! Append the new string and a NewLine
]
```

This sets the variable **out** to the value "`B\nCADAB\n`".

A set of statements may be made into a conditional block using the explicit EXIT statement within the loop.

```
SET x "ABRACADABRA";
SET out "Not Found";
[
       GREP x "A" "R";
       SET out *;          ! OUT gets the result of the GREP
       EXIT;
]
```

This finds the first occurrence of a string bracketed by "A" and "R". If no such string is found, the variable **out** is left with its initial value of "Not Found". In this case, however, **out** is set to "`B`".

`ADD res inc` **Add the increment to the given variable**

XXX	res	I O		
	inc	I		

xxx

xxx

xxx

Examples:

```
    ADD Count 1
```

Note:

xxxxxxxxxxxxxx

`CALL name Param` **Invoke a subprogram**

CALL	name	I	“ ” or “;” delimited pairs	
	Param	I	space delimiter	
	“*”	O		Result is placed in “*”

xxx

xxx

xxx

Examples:

```
    XXX xxx
```

Note:

xxxxxxxxxxxxxx

`EVAL expr` **Evaluate a numeric expression**

EVAL	expr	I O		

xxx

xxx

xxx

Examples:

```
        EVAL "123+456"
```

Note:

xxxxxxxxxxxxxx

`EXEC commandline` **Execute a DOS command line**

EXEC	commandline	I	space-delimited text	Command line

Execute the DOS command line. The first whitespace delimited parameter must be the name of an executable file.

The current directory is the one used by the program.

The path will be searched for the executable, if an explicit path is not specified.

The specified program is executed as an independent process. The interpreter does not wait for completion, and no completion results are available.

Failure to execute the program (file not found, bad path or filename, bad file format, etc.) constitutes a failure which terminates the current loop.

Examples:

```
        EXEC "sol";

        EXEC "C:\\windows\\system32\\sol.exe";
```

Play a game of Solitaire.

Note:

Since many programs assume the “/” character is used as a switch or option delimiter it is not possible for arbitrary replacement of the ‘/’ with the ‘\’ character for file name convenience.

The “C:\\windows\\program” notation is required.

`EXIT` Exit a loop unconditionally

EXIT				

The currently enclosing loop is exited. If not in a loop, the current function is exited.

Used to implement conditionals and straight-through execution for IF-THEN-ELSE style constructs.

In order to prevent the (expected) failure of a statement from terminating the program, enclose it in a “loop” which always exits.

Examples:

```
EXIT;
[ RANGE x 1 10; SET Msg “Valid”; EXIT | Set Msg “Invalid”; EXIT ]
[ LOAD x “Log”; EXIT ]; SET x x “\nRunning”; STORE “Log” x;
```

`FIELD str delim index` **Extract a field from a string**

FIELD	str	I		Fields separated by the given delimiter
	delim	I	delimiter	
	index	I		Index of the field to select
	‘*’	O		Result is returned in ‘*’

The delimiter must be a single character.

Index zero means return the entire string.

Indices beyond the number of fields in the string return NULL.

If the delimiter does not exist in the string, the entire string is presumed to be index one.

Examples:

```
FIELD “A B C” “ ” 0;       ! Result ‘*’ is “A B C”
FIELD “A B C” “ ” 1;       ! Result ‘*’ is “A”
FIELD “A B C” “ ” 2;       ! Result ‘*’ is “B”
FIELD “A B C” “ ” 3;       ! Result ‘*’ is “C”
FIELD “A B C” “ ” 4;       ! Result ‘*’ is “”
FIELD “A B C” “X” 1;       ! Result ‘*’ is “A B C”
```

`FORMAT var codes` **Format a numeric value based on codes**

FORMAT	var	I O	“ ” or “;” delimited pairs	
	codes	I		
		O		Result replaces the input number.

xxx

xxx

xxx

Examples:

```
XXX xxx
```

Note:

xxxxxxxxxxxxxx

`FUNCTION name` **Define the beginning of a subprogram**

FUNCTION	name			
		O		

xxx

xxx

xxx

Examples:

```
XXX xxx
```

Note:

xxxxxxxxxxxxxx

`GREP string Lpatt Rpatt` **Search for text between given delimiters**

GREP	ParamID	I	“ ” or “;” delimited pairs	
	ParamID	I	space delimiter	
	ColorID	I		
		O		Result is xxxx

xxx

xxx

xxx

Examples:

```
XXX xxx
```

Note:

xxxxxxxxxxxxxx

`INPUT` **Wait for keyboard input**

INPUT	ParamID	I	“ ” or “;” delimited pairs	
	ParamID	I	space delimiter	
	ColorID	I		
		O		Result is xxxx

xxx

xxx

xxx

Examples:

```
XXX xxx
```

Note:

xxxxxxxxxxxxxx

`LOAD res filename` **Read a file into a variable**

LOAD	resID	O		contents of file loaded into res
	filenameID	I O	\n delimiter	One or more filenames
	'*'	O		Result is actual filename loaded

xxx

xxx

xxx

Examples:

```
LOAD x "C:\\autoexec.bat";
LOAD x "C:/autoexec.bat";
LOAD x "http://cnn.com";
```

Note:

As a convenience for windows filenames all "/" characters are replaced by "\". This prevents the need to write literals using the "\\" notation. This applies only to disk file I/O; internet URIs are unaffected.

`MAPINIT ParameterID` **Initialize the bitmap for a map.**

MAPINIT	ParameterID	I	" " or ";" delimited pairs ignore commas	Vertical / Horizontal pair for Size, Vertical / Horizontal pair for Scale, and Vertical / Horizontal pair for Center Optional Background FileName
		O		Result is modified bitmap in the system Map

Initializes the map graphic object. <ParameterID> is a string with a series of space-separated parameters for the initialization. The numbers in the parameter represent

<mapHeight> <mapWidth> <scaleHeight> <scaleWidth> <centerNorthing> <centerEasting>

For clarity, the numbers may use commas as thousands separators and multiple spaces or semi-colons may delimit the values.

Examples:

```
MAPINIT "480 640 480000 640000 3630000 285000";
MAPINIT "480; 640;   480,000;  640,000;    3,630,000;  285,000";
MAPINIT "480 640 480000 640000 3630000 285000 Background.bmp";
```

Note:

The UTM zone is not included in the initialization. The map is assumed to lie fully within one zone. The boundary condition with lines that cross zones drawn on a single map is not handled by the current implementation . I have not explored the ramifications of this.

A graphic canvas is initialized to the size given in pixels. This canvas will become a rectangular segment of a map with dimensions given by the scale values. The vertical and horizontal axes may have different scale factors. The center coordinates establish the location of the map in the plane. The scale and center values are integers, thus making UTM coordinates most appropriate for use here.

The optional background FileName allows a previously created bitmap image to be loaded from a disk file. The intention is to have a pre-computed base map which is loaded and then populated with markers and routes as required.

I do not know what happens if the pixel dimensions of the file differ from those specified in the parameter.

If no background FileName is specified the canvas will be filled with the default color clInfoBk (ivory).

Light Gray grid lines and a copyright notice will be drawn on the canvas.

The system supports a single, global map canvas. All operations relating to initialization, drawing lines and marks, and saving the results implicitly refer to the single canvas.

Following initialization with **MAPINIT** a map will normally be drawn using calls to **MAPLINE**. Then locations will be indicated using **MAPMARK**. The resulting file will be saved using a call to **MAPSAVE**.

`MAPLINE CoordListID PenID ColorID` **Draw a poly-line on the map canvas**

MAPLINE	CoordListID	I	“ ” or “;” delimited pairs	Northing / Easting pairs
	PenID	I	space delimiter	width / style
	ColorID	I		Numeric Color value
		O		Result is modified bitmap in Map

The space-delimited list of Northing / Easting pairs in <CoordListID> is used to draw a series of line segments on the map. These segments are scaled to the values give in the previous **MAPINIT** call. For clarity, the numbers may use commas as thousands separators and multiple spaces or semi-colons may delimit the values.

The string in <PenID> consists of a space-delimited pair of numbers representing the line width in pixels and pen style. If unspecified the width will be one pixel and the line will be solid. Dashed and dotted lines are only available in lines one pixel wide.

The number in <ColorID> will normally be a hexadecimal integer representing the blue/green/red intensities.

Examples:

```
MAPLINE "1000 1000; 1000 2000; 2000 2000; 2000 1000"  "1 2"  0xFF00FF
```

Note:

The UTM zone is not included in the coordinates. The map is assumed to lie fully within a single UTM zone. The boundary condition with lines that cross zones drawn on a single map is not handled by the current implementation . I have not explored the ramifications of this.

The system defines some symbolic color names for convenience and clarity.

Name	Value	Name	Value	Name	Value
$black	0x000000	$red	0x0000FF	$magenta	0xFF00FF
		$green	0x00FF00	$cyan	0xFFFF00
$white	0xFFFFFF	$blue	0xFF0000	$yellow	0x00FFFF

MAPLINE draws a single, connected polyline. Multiple calls are used to create disjoint lines in different colors, widths and styles.

`MAPMARK CoordListID MarkID ColorID` **Mark a set of locations on the map bitmap.**

MAPMARK	CoordListID	I	" " or ";" delimited pairs	Northing / Easting pairs
	MarkID	I		binary character string of bits
	ColorID	I		Numeric Color value
		O		Result is modified bitmap in Map

The space-delimited list of Northing / Easting pairs in <CoordListID> is used to draw a series of glyphs on the map. These locations are scaled to the values give in the MAPINIT call. For clarity, the numbers may use commas as thousands separators and multiple spaces or semi-colons may delimit the values.

The string given by <MarkID> represents bits in a square glyph. Each character in the string represents eight bits. The dimensions of the glyph are square and equal to the square root of eight times the number of characters in the string. One bits are rendered as the specified color on the map, zero bits are left unchanged and are thus transparent. The center of the glyph is placed on the map at the specified coordinates.

The integer given by <ColorID> represents the blue/green/red intensities of the glyph.

The system defines some symbolic glyph names for convenience and clarity.

$square8	$box8	$circle8	$dot8	$diamond8
$square16	$box16	$circle16	$dot16	$diamond16

For example, $box8 is defined as "\FF\81\81\81\81\81\81\FF".

`MAPSAVE FileNameID` — **Write the contents of the map bitmap to a graphic file.**

MAPSave	FileNameID	I		Windows Disk File name
		O		Result is disk file containing the map
	'*'	O		Actual disk file name

The current map graphic is saved to the specified disk file.

The default file format is BMP. The file extension must be specified.

If the specified file extension is ".jpg" the format will be a JPEG image. Anything else will be a BMP.

For convenience, all '/' characters are converted to '\'. This function only writes to disk files; URIs are not allowed.

Example:

```
MapInit "250 500 500 1000 1000 1000";
MapLine "1000 1000 1100 1200" "1 0" $black;
MapMark "900 900" $Square16 $red;
MapMark "1100 1100" $Box16 $green;
MapMark "900 1100" $Diamond16 $blue;
MapMark "1100 900" $Circle16;
MapMark "1100 950" $Dot16;
MapMark "1000 850" $Dot16 $magenta;
MapMark "1000 900" $Dot16 $cyan;
MapMark "1000 950" $Dot16 $yellow;
MapSave "E:/Junk.bmp";
```

`MAPTOUR res coords limits` — Reorder coordinate points into a path

MAPTOUR	res	O	" " or ";" delimited pairs	
	coords	I	space delimiter	
	limits	I		
		O		Result is xxxx

xxx

xxx

xxx

Examples:

```
XXX xxx
```

Note:

xxxxxxxxxxxxxx

`MINMAX res values` **Description**

MINMAX	res	O		
	values	I	“ ” or “;” delimited values	

xxx

xxx

xxx

Examples:

```
        MINMAX x "1 2 3 4"
```

Note:

xxxxxxxxxxxxxx

`OPEN filenames` **Display Open File Dialog to get filename list**

OPEN	filenames	I O	“\n” delimited names	

xxx

xxx

xxx

Examples:

```
        XXX xxx
```

Note:

xxxxxxxxxxxxxx

`POST URL text` **Send specified text to a URL**

POST	URL	I		
	text	I		
	$HEADER	O		
	$BODY	O		
	"*"	O		Result is xxxx

xxx

xxx

xxx

Examples:

```
    XXX xxx
```

Note:

xxxxxxxxxxxxxx

`PRINT text` **Print the text on the system default printer**

PRINT	text	I	"\n" delimited lines	

xxx

xxx

xxx

Examples:

```
    XXX xxx
```

Note:

xxxxxxxxxxxxxx

`RANGE value lowlim highlim` **Validate that the value is in a specified range**

RANGE	value	I		
	lowlim	I		
	highlim	I		

The Value is checked against the limits. If it is not within the specified bounds the statement fails. Failure causes the current loop to be exited, or the ELSE section to be executed.

If all three parameters contain only numeric characters (‘0'..’9',’-‘) the comparison will be based on converted integers. Otherwise the comparison is an ASCII string comparison.

Examples:

```
[ ADD Count 1; RANGE Count 1 10; SET List List Count “ ”; ]
[ ADD Count 1; RANGE Count 1 10; SET List List Count “ ”;
       | SET List List “Done.”; EXIT; ]
```

Note:

xxxxxxxxxxxxxx

`REP str old new` **Replace all occurrences of old with new in str.**

REP	str	I O		
	old	I		
	new	I		

xxx

xxx

xxx

Examples:

```
XXX xxx
```

Note:

xxxxxxxxxxxxxx

`SELECT ID XML pattern` **Description**

SELECT	ID	I	" " or ";" delimited pairs	
	XML	I O		
	pattern	I		
	"*"	O		Result is xxxx

xxx

xxx

xxx

Examples:

```
XXX xxx
```

Note:

xxxxxxxxxxxxxx

`SET res str str ...` **Concatenate strings and set the value of the result**

SET	res	O		Result is concatenated strings.
	str	I		

This is the assignment statement. Zero or more strings are concatenated to form a value which is assigned to the string res.

Examples:

```
SET Count 0;

SET Null;

SET Msg "Hello" " " "World.";


[ ADD Count 1; RANGE Count 1 10; SET List List Count " "; ]
```

Note:

This command allows a variable number of input parameters.

The variable used as the result may be used as an input string without complications. This allows pre-pending, appending and duplicating text to occur in a natural manner.

`SHOW str` **Display the string in status or on the user display**

SHOW	str	I	“ ” or “;” delimited pairs	
	$STATUS	O		Result is xxxx

xxx

xxx

xxx

Examples:

```
    XXX xxx
```

Note:

xxxxxxxxxxxxxx

`STORE filename str` **Store the string in the given file**

STORE	filename	I	filename	
	str	I	body of file to write	
	"*"	O		Result is actual filename

The filename may be a URI beginning with "ftp:" or "http:", or it may be a disk file name.

In order to access a FTP site, the username and password must be placed into $user and $pass.

xxx

Examples:

```
SET Text "Hello, World.";
SET $user "guest"; SET $pass "";
STORE "ftp://hostname/path/file" Text;
STORE "http://hostname/path/file" Text;
STORE "C:\\Temp\\File.txt" Text;
STORE "C:/Temp/File.txt" Text;
```

Note:

As a convenience forward slashes "/" are allowed in windows disk file names. They are replaced by backslashes "\" to eliminate the need to use the escaped "\\" in literals.

`SUBSTR str patt resA resB ...` **Divide the string into fields based on a pattern**

SUBSTR	str	I		
	patt	I		
	resA	O		
	resB	O		Result is xxxx

xxx

xxx

xxx

Examples:

```
XXX xxx
```

Note:

xxxxxxxxxxxxxx

`TAGS TagList XMLtext` **Validate an XML file and extract its tags**

TAGS	TagList	O		
	XMLtext	I		

xxx

xxx

xxx

Examples:

```
XXX xxx
```

Note:

xxxxxxxxxxxxxx

`UTM res str` **Convert Lat / Lon to UTM and vice versa**

UTM	res	O		
	str	I		

xxx

xxx

xxx

Examples:

```
XXX xxx
```

Note:

xxxxxxxxxxxxxx

Program Examples

```
!Test the loop termination conditions for grep.
set ls "This is a test"
[ grep ls "This i" "s a test"; show "Did not exit." show z; add z 1]

!Test Looping and Range check conditionals
set ls " a, b, c, d, e, ";
set lx "In ";
[grep ls " " ",";
  [range * "b" "d"; set lx lx *; show lx; exit ]
]
set ls " a, b, c, d, e, ";
set lx lx ", Out ";
[grep ls " " ",";
  [range * "d" "b"; set lx lx *; show lx; exit; ]
]
set ls " 1, 2, 3, 4, 5, ";
set lx lx ", In ";
[grep ls  " " ",";
  [range * "2" "4"; set lx lx *; show lx; exit ]
]
set ls " 1, 2, 3, 4, 5, ";
set lx lx ", Out ";
[grep ls " " ",";
  [range * "4" "2"; set lx lx *; show lx; exit; ]
]
show lx;


!Test the SUBSTR implementation that breaks a string into
!multiple named strings
set LS "This is a test of the emergency broadcast system. Beep.";
substr LS "AAAA.BB.C.AAAA" LA LB;
show LA;
```

```
! This downloads all the county files for Texas from the Tiger Server
! We know that the FIPS codes for TX counties are odd and have no gaps.
set Num 1;
[
  format Num "ZZZ";  show Num;
  set Name "http://www.census.gov/geo/www/tiger/tigerua/TX/TGR48" Num ".ZIP";
  load x Name;
  set Name "E:/TigerMapData/TGR48" Num ".zip";
  store Name $BODY;
  add Num 2;
]
```

```
! This is loading real Tiger records and looking at them
! It creates a complete county road feature file with type 1 and 2 records
! To be useful, the file needs to be sorted.
! We also include water features.
load x "E:/TigerMapData/TGR48499.RT1";  ! Wood County Line Features
load y "E:/TigerMapData/TGR48499.RT2";  ! Wood County Shape Features
set x "\n" x "\nend";  ! Need bracketing delimiters for grep...

set Pattern "IIIIIAAAAAAAAAA....BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBCCCC"
            "..DDD................."
            "......................................."
            "................................."
            "......................................."
            "EEEEEEEEEEFFFFFFFFFGGGGGGGGGGHHHHHHHHH";

[
  grep x "\n" "\n";
  set Line *;
  substr Line Pattern TLID FENAME FETYPE CFCC FRLONG FRLAT TOLONG TOLAT TVER;
  [ range CFCC "A00" "H99"; range CFCC "H" "A999";
    set out out TVER TLID FENAME FETYPE CFCC FRLONG FRLAT TOLONG TOLAT "\n";
    add count 1;
    exit;
  ]
  add total 1;
  set msg count " of " total;
  show msg;
]
set out out y;  ! Append the shape features.  This can be sorted by column 6
                ! to 15 to group the chains
store "E:/TGR48499.txt" out;
exit;
```